HONORABLE JAMES L. ROBART

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WASHINGTON
AT SEATTLE

REC SOFTWARE USA, INC.,

Plaintiff,

vs.

BAMBOO SOLUTIONS CORPORATION; MICROSOFT CORPORATION; SAP AMERICA, INC.; and SAP AG,

Defendants

Case No. 2:11-cv-554-JLR

MICROSOFT CORPORATION'S OPENING CLAIM CONSTRUCTION BRIEF

LAW OFFICES
DANIELSON HARRIGAN LEYH & TOLLEFSON LLP
999 THIRD AVENUE, SUITE 4400
SEATTLE, WASHINGTON 98104
TEL, (206) 623-1700 FAX, (206) 623-8717

TABLE OF CONTENTS

Table of Authorities

Page(s)

CASES

I. INTRODUCTION

As the Court is aware, the intrinsic record of a patent memorializes a bargain struck between the applicant and the public: in exchange for a temporary monopoly, the applicant clearly defines the metes and bounds of his alleged invention. The applicant does this both through the specification and claims themselves and through representations to the Patent Office. Microsoft Corporation's ("Microsoft") proposed constructions hold the patentee to his bargain and are consistent not only with the intrinsic record, but also with the patentee's 10-year representation of the scope of his alleged invention on his public web site.

REC Software USA, Inc.'s ("REC") constructions exemplify the amnesia that a patentee must have when, after nearly 20 years of representations to the public that the alleged invention is of a certain scope, it suddenly argues the scope is much broader. REC would like this Court and Microsoft to ignore representations it made to the Patent Office and the public. REC's constructions not only contradict the alleged inventor's statements but are inconsistent with *Markman v. Westview Instruments*, *Phillips v. AWH Corp.*, and their progeny.

II. BACKGROUND OF THE TECHNOLOGY AND ALLEGED INVENTION

A. Technological Background

U.S. Patent No. 5,854,936 ("the '936 patent"), the only asserted patent in this suit, relates to the behind-the-scenes work done by a computer's operating system to prepare complex computer programs for execution on the computer. Both in the patent specification and during prosecution, the patentee explained that the alleged invention operates in conjunction with an operating system or environment that dynamically links multi-module programs prior to execution. This dynamic linking allows a single copy of a module to be shared among multiple programs, saving storage space, and also supports the separate development and maintenance of the modules.[1] The ’936 patent claims certain improvements to the dynamic linking process.

B. The Alleged Invention vs. the Prior Art

The ’936 patent describes an alleged improvement related to how a user’s[2] operating system[3] prepares for execution what the patentee termed “multi-module” computer programs—an example of which is shown in Figure 2 of the patent. However, the patentee does not claim to have invented multi-module programs or any technique for modules to refer to each other—on the contrary, the patentee consistently acknowledged such to be in the prior art.[4]

As described in further detail in the individual constructions, the patentee extensively explained to the PTO and the public how prior art operating systems (e.g., Windows 3.x and OS/2) prepared dynamically linked multi-module programs for execution.[5] The applicant coined the term “forming an association” to describe that process, and the patentee’s admissions during

[1] JCC Appendix B, Exhibit 13 (D.I. 112-18) at 1 (definition of “dynamic linking”).
[2] The specification defines “users” as computer terminals or other computer systems. *See* ’936 patent at 3:5-6.
[3] U.S. Patent No. 5,649,204 (“the ’204 patent”), is the parent of the asserted ’936 patent. Its claims recite “operating system program,” while the ’936 patent claims merely recite “first program.” For brevity and clarity, this description uses the term “operating system” as shorthand for the portion of the system playing the role claimed as “operating system program” (’204 patent) or “first program” (’936 patent).
[4] *See, e.g.*, JCC Appendix B, Exhibit 6 (D.I. 112-11 (’204 FH Paper No. 8, 8/18/94 Amdt.)) at 4-5 (“The present invention is directed to a code server for a computer system **that operates in an environment which processes coded programs in discrete modules which may have embedded references to other discrete modules** (multi-module system)” (emphasis added).).
[5] *See, e.g.*, JCC Appendix B, Exhibit 7 (D.I. 112-12 (’204 FH Paper No. 11, Appellant’s Br.)) at 8 (“Forming an ‘association’ for a multi-module program involves the dynamic linking of discrete modules together and this traditionally requires a two-stage process. The first stage involves the operating system inspecting a code module for embedded references to other discrete modules. If the operating system does not already possess the linkage information for the referenced code modules in its linkage table, then it initiates a search for those unresolved code modules and extracts linkage information from each…. This searching and locating process continues recursively until the operating system has formed a complete association and has created entries regarding the individual code modules for its linkage table.”).

and after prosecution confirm this point.[6] The patentee described the alleged invention as a way to assist the operating system in gathering the necessary linkage information so the operating system could "form the association" of a program more efficiently.

The problem with prior art systems, according to the patentee, was that the process of searching for modules (on a computer and/or across a network) and extracting the "linkage information" needed for dynamic linking took time, and was repeated anew and separately by each computer executing a given multi-module program. In response to the purported problem, the specification proposes adding to the system a "code server" that allegedly streamlines this aspect of the above-described process of preparing a multi-module program for execution. While "forming an association" would still be done by the operating system, the code server would take over the task of finding the necessary code modules, so the operating system need only make a single inquiry (to the code server) for a code module. The code server also would store the linkage information in its "module information table" so that the next time an operating system requested the same module, the code server could respond with the linkage information from the table rather than starting the search process anew.[7] Further illustration of the code server is shown in Penner Decl., Exhibit 2, which describes Figure 3.

[6] *See, e.g.*, JCC Appendix B, Exhibit 6 (D.I. 112-11 ('204 FH Paper No. 8, 8/18/94 Amdt.)) at 6 ("As used in the specification and claims, applicant refers to the creation by the operating system of the dynamic link tables necessary for the execution of a multi-mode program as 'association'.").

[7] *See, e.g.*, JCC Appendix B, Exhibit 5 (D.I. 112-10 ('204 FH Paper No. No 6, 3/1/94 Amdt.)) at 4-5 ("Once the module information has been located the first time by the code server, an operating system that later requests the same module information from the code server that is now contained in the code server's module information table is quickly retrieved by reading the module information therein, thereby eliminating the time-consuming task of searching the network. The code server therefore provides a central server for the distribution and storage of code module information.").

III. ARGUMENT

A. The Asserted Patent Claims

REC has alleged infringement of claims 1, 2, 4, 5, 6, 8, 9, 10, 15, 17, 18, and 22 of the ’936 patent.[8]

B. Principles of Claim Construction

“To ascertain the meaning of claims, [the court] consider[s] three sources: The claims, the specification, and the prosecution history.”[9] These three sources are the intrinsic evidence; public records available for all to rely upon when determining the meaning of a patent.[10]

Claim interpretation begins with the actual language of the claims.[11] Generally, the words, phrases and terms in patent claims should receive their ordinary and accustomed meaning.[12] If the ordinary meaning is not apparent from its use in the claim, the court looks to the specification to provide meaning.[13] The specification acts as a “concordance” for claim terms, and is thus the best source beyond claim language for understanding claim terms.[14]

The inventor is free to use the specification to define claim terms as she wishes, and the court must defer to an inventor’s definition, even if it is merely implicit in the specification.[15] If the patentee’s definition conflicts with the plain and ordinary meaning, the patentee’s

[8] For the Court’s reference, Microsoft provides a chart setting forth asserted independent claims 1 and 9, with the disputed terms emphasized. *See* Declaration of Scott A. Penner in Support of Microsoft Corporation’s Opening Claim Construction Brief (“Penner Decl.”), Exhibit 1 at 1.
[9] *Markman v. Westview Instruments, Inc.*, 52 F.3d 967, 979 (Fed. Cir. 1995) (citations omitted).
[10] *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1583 (Fed. Cir. 1996).
[11] *Bell Commc’ns Research, Inc. v. Vitalink Commc’ns Corp.*, 55 F.3d 615, 619-20 (Fed. Cir. 1995).
[12] *Johnson Worldwide Assocs. v. Zebco Corp.*, 175 F.3d 985, 989 (Fed. Cir. 1999) (noting a “heavy presumption in favor of the ordinary meaning of claim language”).
[13] *Id.* at 990.
[14] *Phillips v. AWH Corp.*, 415 F.3d 1303, 1315 (Fed. Cir. 2005).
[15] *Id.* at 1316, 1320-21 (noting that a court cannot ignore explicit or implicit definitions).

construction is controlling.[16] This is particularly true when a patentee coins a new term, which has no customary meaning outside of the patent.[17] Likewise, if a patentee ascribes a particular meaning to a term in order to obtain a claim's allowance during patent prosecution, he cannot attempt to broaden the definition of that term in litigation.[18]

C. The Disputed Claim Terms

The parties disagree on the constructions of a total of twelve claim terms. The parties agree that four terms are in means-plus-function form and thus governed by 35 U.S.C. § 112 ¶ 6, and that seven terms are not in means-plus-function form. The parties disagree whether the twelfth term is in means-plus-function form.[19]

1. <u>Terms not governed by 35 U.S.C. § 112 ¶ 6</u>

a. "first program that is executing on a computer"; "second multi-module program"

The dispute surrounding these two terms is whether the word "program," as used by the patentee, requires construction. Microsoft contends it does not. The word's usage in the claims comports with its well-understood meaning in the computer science field, and the specification gives no indication at all that the patentee intended for "program" to have a special meaning.

REC proposes to replace the eleven (11) words of these two terms with ninety-eight (98) words of constructions. Over half of the words do nothing but unnecessarily belabor a fundamental principle of claim drafting—that the "first" program is different from the "second" program. REC's constructions are lengthy and would be confusing to a jury. Microsoft

[16] *Honeywell Int'l, Inc. v. Universal Avionics Sys. Corp.*, 493 F.3d 1358, 1361 (Fed. Cir. 2007); *C.R. Bard, Inc. v. U.S. Surgical Corp.*, 388 F.3d 858, 863 (Fed. Cir. 2004).
[17] *See, e.g., MyMail, Ltd. v. Am. Online, Inc.*, 476 F.3d 1372, 1376 (Fed. Cir. 2007); *CCS Fitness, Inc. v. Brunswick Corp.*, 288 F.3d 1359, 1366 (Fed. Cir. 2002).
[18] *See, e.g., Computer Docking Station Corp. v. Dell, Inc.*, 519 F.3d 1366, 1374 (Fed. Cir. 2008).
[19] For the Court's convenience, Microsoft provides a side-by-side comparisons of the parties' constructions for each disputed term. *See* Penner Decl., Exhibit 1 at 2-6.

contends that ordinary usage of the word "program" in a computer software patent is what the patentee clearly intended and thus it should be given its plain and ordinary meaning.

b. "code server"

The parties' dispute with respect to code server is whether the code server must be a "disjoined task (separate process from the first program)" or merely "computer software" that "is an identifiable set of computer instructions other than the first program." Microsoft's proposed construction traces the language of the specification while REC's construction overly broad, vague, and reads out a disclosed embodiment.[20]

First, the term "code server" is not a term typically used by those skilled in the art; it was coined by the patentee and thus its meaning must be derived from the intrinsic record. Microsoft's proposed construction is taken directly from the specification. The patentee defined the code server as "embodied in a transaction oriented protocol for communication between the user and the code server 18 which allows the users 12, 14, and 16 and the code server 18 to be on two remote computer systems or to be configured as *separate disjoined tasks* in a multi-processing system on a single computer." '936 patent at 3:21-26.[21] Because separate tasks in a multi-processing system are known as processes, the patentee defined the code server as at least being a separate process from the requesting program (i.e. the first program).

Consistent with this, the patentee explained how the code server was isolated from the requesting program. That is, the specification teaches that the code server's functionality is

[20] The phrase "identifiable set of computer instructions other than the first program" in REC's construction appears nowhere in the specification and remains ambiguous in that it is unclear whether the identifiable set must be completely disjoint from the first program, thus defeating the purpose of claim construction. Moreover, REC's limitation of the "code server" to "software" ignores that the specification discusses implemented the code server in hardware. '936 patent at 3:118-19 ("The code server 18 may be a separate outboard piece of hardware.").
[21] As described in fn. 2, *supra*, the patent defines "users" as computers. The operating system/first program operates on these "user" computers.

isolated from the first program by use of a "protocol receiver" and a "protocol sender." *See id.* at 3:28-40. By using the protocol sender, the requesting program can be isolated from and ignorant of the code server process. *Id.* The figures further confirm this point. Figure 1 shows the protocol receiver (20) and protocol sender (22) as being part of the code server, which is isolated from the requesting program (represented by the dotted lines around box 18). Figure 3 describes the use of the protocol receiver and sender by showing the user isolated from the code server (again showing the dotted line around box 18) and the user making a request that is "received" in box 3, described in the specification as the protocol receiver. *See id* at 4:37-38.

In other words, the specification and figures paint a consistent story: the code server either resides on a separate machine or must be completely unrelated (a "disjoined task") from the requesting entity (i.e. the first program). Microsoft's construction uses the exact language of the specification and covers both identified scenarios. However, the phrase "disjoined task" may be confusing to a jury. In order to make it more understandable, and to accurately reflect what one of skill in the art would have understood the term "disjoined task" to mean, Microsoft has added "separate process from the first program" in parenthesis for clarity.

The understanding that a "disjoined task" is a "separate process from the first program" is confirmed by the file history. There the patentee first explained that the "code server…*processes* coded programs."[22] In other words, the code server must be executable and must be capable of *processing* information. The patentee also explained the isolation of the code server from the requesting program by noting that the code server's information was specifically maintained by the code server itself.[23]

[22] JCC Appendix B, Exhibit 5 (D.I. 112-10 ('204 FH Paper No. No 6, 3/1/94 Amdt.)) at 4.
[23] *Id.* at 4 ("The code server *maintains its own set* of module information *independent of* that which *the operating system* creates between various code modules for forming an association.").

Further, Mr. Pickett's public explanation of how the code server worked is consistent with Microsoft's explanation. Mr. Pickett's website explained: (1) "code server" was a term invented by the patentee;[24] (2) a code server must be a disjoined task (separate) from the requesting entity;[25] and (3) the code server is its own program (process).[26]

The intrinsic evidence, file history, and Mr. Pickett's own public declaration about the meaning of "code server" are all consistent with Microsoft's definition. The code server must possess the attribute of being a "disjoined task (separate process from the first program)" otherwise it would inconsistent with the purpose of the invention, as described above in Section I.B., which was to have the "code server" performing tasks that would have otherwise been done by the operating system.

c. "module information"

The dispute between the parties is whether "module information" can be limited to a subset of what a module links to (REC's proposal), or if it must be the complete set of linkage information for the module (Microsoft's proposal).

As an initial matter, the claims help to define the scope of what is meant by "module information." For example, independent claim 1 identifies several properties of the "module information." The claims note that module information has four attributes: (1) it is stored in the

[24] *See* JCC Appendix B, Exhibit 9 (D.I. 112-14) at 1 ("Code Servers are REC Software's patented invention. We developed the idea back in 1990.").

[25] *See id.* at upper right figure; JCC Appendix B, Exhibit 10 (D.I. 112-15) at 1 (showing requesting entity (client side) and the code server (server side) as being separated by the dotted line); JCC Appendix B, Exhibit 13 (D.I. 112-18) at 2 (defining operating system (OS) as "a program that loads other programs and provides them with a common set of services. *Addition of a Code Server to an OS* separates part of the linker-loader so that it can be used across the network."). In other words, the code server is *not* part of the operating system.

[26] *See* JCC Appendix B, Exhibit 13 (D.I. 112-18) at 2 (defining "protocol" as "[t]he method by which *two programs* or logical entities (e.g. Code Servers) *communicate* and work together. *In the context of the code server patents, the information transmitted by protocol is platform-*

module information table; (2) it is searched for by the code server; (3) it is read by the code server; and (4) it is provided back to the first program in order to allow the association to be formed. Given these attributes, the specification teaches that module information is the sum total of the data stored in the code server that gets returned with a single request.

As described above, Figure 2 represents "the way in which a typical program might be configured in modules in order to run in a multi-module operating system." '936 patent at 4:10-12. The specification explains that what is represented in Figure 2 is the "linkage information." *Id.* at 3:14-15. So information representing that the Main Module links to Modules A, B, and C and that module A links to the Main Module and Modules D and E, and so forth as shown in Figure 2. The patent makes this linkage clear by teaching that, "*[c]ollectively* the linkage information [set forth in Figure 2] forms an associative set for the main code module and code modules A through E…*It is this type of associative information that is stored in the code module information table* 24 in FIG. 1." *Id.* at 4:25-31 (emphases added). This is further confirmed by the specification which explains that when a user makes a request "for a particular code module" the code server "will immediately locate *all of the associative information pertaining to that module in the code module information table*." *Id.* at 4:31-34 (emphasis added). Therefore, the module information, which is what gets stored in the code module information table, consist of the collective set of associative information, not subsets of that information.

Microsoft's construction comports with the specification and figures. REC's construction, on the other hand, which is limited to "one or more other modules" would encompass a situation where a request for "Main Module" of Figure 2 returned only "Module B" for example. In such a circumstance, the code server would fail its intended purpose. After

---

independent, and must not contain address references"). In other words, the code server must be a separate program or a separate logical entity (like a separate piece of hardware).

receiving only information about Module B, the system would have no information about Modules A, C, D, or E and thus would not be able to form the association. Or, if the code server had to then make additional requests for Modules A, C, D or E, the system would be in violation of the specification's teaching that *all of the associative information* is *immediately* returned.

d. "searching a module information table for module information in response to said request associated with said discrete module"; "a response that includes module information associated with said discrete module"

The parties agree that the code server: (1) receives from the first program a request associated with a discrete module; (2) searches its module information table in response to the request; and (3) sends a response to the first program containing the results of the search. The common dispute in the above two terms is the scope of the search and subsequent response: whether it need involve only the smallest shred of information that might be argued to pertain to the discrete module (REC's constructions), or must include <u>all information</u> contained in the table pertaining to the discrete module—the information that the "first program" needs to perform its task of preparing the multi-module program for execution (Microsoft's constructions). Both the patent itself and the patentee's admissions confirm that Microsoft's constructions are correct.

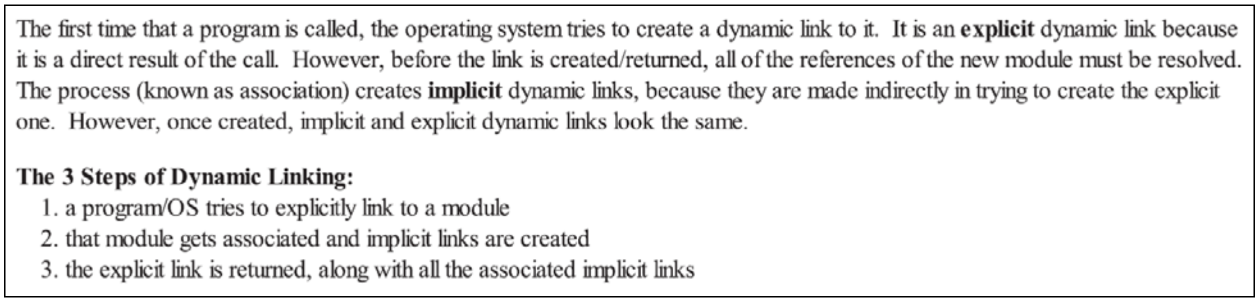
First, Microsoft's constructions comport with the specification's description of "the invention":[27] "A request by a user for a particular code module will immediately locate ***all of the associative information pertaining to that module*** in the code module information table 24 if it has been previously stored there." '936 patent at 4:31-35 (emphasis added). And for good reason—the purpose of the request is to receive information required to prepare the multi-

[27] The Federal Circuit has held that in circumstances where the patentee consistently refers to the disclosed invention as a single invention rather than a preferred embodiment (e.g., "this invention," "the present invention"), it can be appropriate to limit the claims to that disclosure. *See, e.g., Honeywell Int'l, Inc. v. ITT Indus., Inc.*, 452 F.3d 1312 (Fed. Cir. 2006) ("The public is

module program for execution. The stated purpose of the invention is to relieve the first program of the task of searching for modules and extracting linkage information. Responding with only a portion of the associative information stored in the module information table would be pointless.

Furthermore, the patent states that "[t]he code server maintains linkage information between the various code modules forming an association representing all the linkage data for the entire program," and that "[o]nce the associative data has been gathered, a coded program may be retrieved quickly and efficiently without the need for repetitive on-line searching because the user need refer only to the code server which contains a look-up tables [sic] storing the data representing the associative information." *Id.* at Abstract. If the first program (user) is to "need refer only to the code server," the code server must send the first program all of the information necessary to make use of the requested module. Returning merely a portion of the required "module information" would not give the first program the information it needs to prepare the program for execution, and thus REC's construction cannot be correct.

The inventor's representations to the public further confirm Microsoft's construction:


The first time that a program is called, the operating system tries to create a dynamic link to it. It is an **explicit** dynamic link because it is a direct result of the call. However, before the link is created/returned, all of the references of the new module must be resolved. The process (known as association) creates **implicit** dynamic links, because they are made indirectly in trying to create the explicit one. However, once created, implicit and explicit dynamic links look the same.

The 3 Steps of Dynamic Linking:
1. a program/OS tries to explicitly link to a module
2. that module gets associated and implicit links are created
3. the explicit link is returned, along with all the associated implicit links

JCC Appendix B, Exhibit 12 (D.I. 112-17) at 2. The inventor described the linkage information to a module (e.g., module A in Figure 2) as an explicit dynamic link, and linkage information between the module and any of its dependent modules (e.g., modules D and E in Fig. 2) as implicit dynamic links. The inventor went on to say in step 3 of dynamic linking that "the

---

entitled to take the patentee at his word and the word was that the invention is [that which was explicit link is returned, along with all the associated implicit links." (emphasis added). For example, in Figure 2 a request to the code server for module A would return linkage information to modules A, D, and E. By so doing, all the dependencies for module A have been resolved and thus module A will work.

REC's constructions ignore the unity of the inventor's disclosure of the purpose and operation of the alleged invention, both in the patent and in his public statements. Moreover, REC's constructions place no meaningful limitations on the claim: coupled with REC's vacuous construction for "module information," REC would be free to argue that "module information... pertaining to the discrete module" could be fulfilled by a single piece of data such as a dependent module's name, which would be insufficient for the alleged invention to work.

e. "forming an association of said multi-module program by said first program"

"Forming an association" is not defined in the '936 patent specification. To be sure, the phrase or variants thereof appear repeatedly, but always in a matter-of-fact way that suggests the reader should understand its meaning. REC doubtless will point to fragments of the specification and declare that some combination of those fragments implicitly describe the process of "forming an association," but the 15-year record of the prosecution history and the patentee's representations to the public estop REC from now adopting such litigation-driven positions.

The place to settle the matter is the file history of the parent '204 patent, which contains the same arguments. First, it was made clear that any novelty of the "code server" invention was only in the context of systems that prepared multi-module programs for execution by "forming an association." Second, it was explained what "forming an association" both is and is not.

disclosed in the specification].").

The Examiner first rejected the claims as unpatentable over U.S. Patent No. 4,941,084 to Terada et al. ("Terada").[28] Mr. Pickett responded with a four-page argument distinguishing Terada. His opening statement clearly and unequivocally framed the discussion to follow:

> **The present invention is directed to** a code server system that operates in an environment which processes coded programs in discrete modules which may have embedded references to other discrete modules (multi-module system)…[29]

Mr. Pickett thus made clear to the Examiner that his alleged invention was only considered novel within a specific environment, which he described as follows:

> Dynamic linking of multi-module systems traditionally refers to a two-step process. Stage 1 involves the operating system inspecting a code module for embedded references to other discrete modules. If the operating system does not already possess the linkage information for the referenced code modules in its linkage table, it will initiate a search for those unresolved code modules and extracting linkage information from each. This process continues recursively until the operating system has formed a complete association and has created entries regarding the individual code modules and its linkage table.[30]

Finally, Mr. Pickett distinguished Terada:

> Claim 9, which is the sole independent method claim [issued claim 1], and claim 17, the sole independent apparatus claim [issued claim 9], are both directed to a system for providing information in a <u>protocol response to an operating system program and then forming an association of a multi-module program in the operating system</u>. Terada does not teach nor suggest how one could use his distributive processing system for assisting in the formation of an <u>association of a multi-module program</u> in the operating system which is required for dynamic linking of multi-module systems.[31]

Two key takeaways from this amendment: First, Mr. Pickett admitted that "forming an association" was not an inventive element of his code server, but rather was a required activity of

[28] JCC Appendix B, Exhibit 5 (D.I. 112-10 ('204 FH Paper No. 6, 3/1/94 Amdt.)) at 6.
[29] *Id.* at 4 (emphasis added).
[30] *Id.* at 5.
[31] *Id.* at 6-7 (emphasis in original).

the multi-module operating system with which the code server interacted. Second, Mr. Pickett described the process of "forming an association" as a required element of "dynamic linking."

Presumably after conducting a search for references disclosing "dynamic linking," the Examiner rejected the claims once again, this time as unpatentable over Terada in light of Tennant, H., "Linking and Loading", Bits & Pieces, 1979 ("Tennant").[32] Mr. Pickett needed to distinguish between his explanation of "dynamic linking" (which required the claimed "forming an association") and Tennant's "dynamic linking." He thus argued that his "dynamic linking" was a special kind:

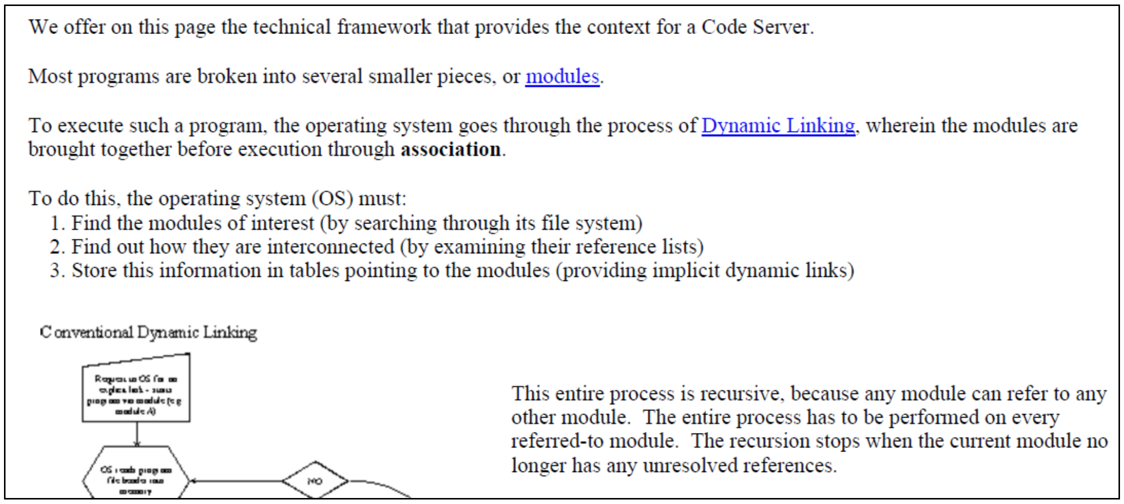
> **Dynamic links which are used in conjunction with forming an <u>association</u>**, as that term is defined by the present invention and used in applicant's specification and claims, generally refer to linkages used in operating systems such as OS/2 from IBM or Windows 3.x from Microsoft, for the loading and execution of code modules.... The code modules, unlike the process of linking, do not address the other code modules directly, **but instead address internal tables built within the operating system itself, i.e., tables of dynamic links**. These dynamic link tables act under control of the operating system as an intermediary addressing the other code modules…. **As used in the specification and claims, applicant refers to the creation by the operating system of the dynamic link tables necessary for the execution of a multi-mode program as "association".** In contrast, the process of linking and loading code pieces, as described by Tennant, is not what applicant's invention refers to as association and Tennant's process is directed to a system that is much more limited in nature.[33]

Mr. Pickett repeated this same argument during the appeal to the BPAI, clearly demonstrating it was not inadvertent:[34] his coined term "association" referred to the operating

[32] JCC Appendix B, Exhibit 6 (D.I. 112-11 ('204 FH Paper No. 8, 8/18/94 Amdt.) at 2.
[33] *Id.* at 5-6 (underlining in original; boldface added).
[34] *See, e.g., Spring Window Fashions v. Novo Industries*, 323 F.3d 989, 996 (Fed. Cir. 2003) ("[T]here is no indication that the detailed distinction of [the prior art] was simply an inadvertent misstatement by the prosecuting attorney for which the applicant should be given a mulligan. The statements distinguishing [the prior art] were detailed, consistent, and repeated. A reasonable competitor would have believed that the applicant's disclaiming statements were not a mere mistake.").

system's creation of <u>tables of dynamic links</u>, and required that the operating system create <u>all dynamic links necessary for the execution of the program</u>.[35]

To erase any vestiges of doubt, after allowance of both the parent '204 patent and the asserted child '936 patent, Mr. Pickett went on to tell the public the same story for the next decade:



We offer on this page the technical framework that provides the context for a Code Server.

Most programs are broken into several smaller pieces, or modules.

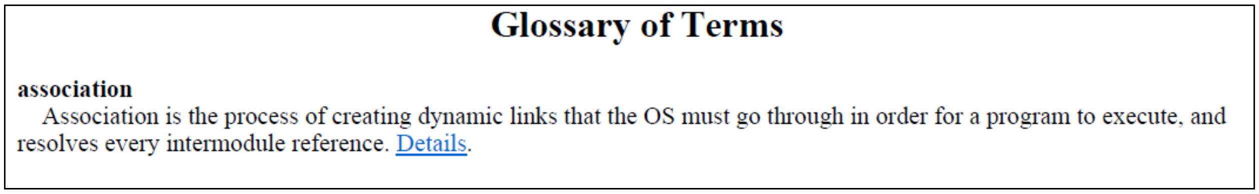
To execute such a program, the operating system goes through the process of Dynamic Linking, wherein the modules are brought together before execution through **association**.

To do this, the operating system (OS) must:
1. Find the modules of interest (by searching through its file system)
2. Find out how they are interconnected (by examining their reference lists)
3. Store this information in tables pointing to the modules (providing implicit dynamic links)

Conventional Dynamic Linking

This entire process is recursive, because any module can refer to any other module. The entire process has to be performed on every referred-to module. The recursion stops when the current module no longer has any unresolved references.

JCC Appendix B, Exhibit 9 (D.I. 112-14) at 1. Mr. Pickett states that the above description is "the technical framework that provides the context for a Code Server," and that "association" referred to the operating system's process of dynamic linking. Helpfully, Mr. Pickett provided a Glossary of Terms on his web site:



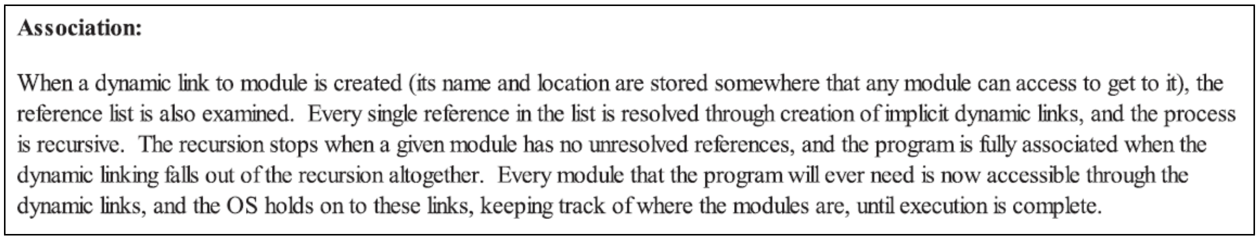
Glossary of Terms

association
Association is the process of creating dynamic links that the OS must go through in order for a program to execute, and resolves every intermodule reference. Details.

JCC Appendix B, Exhibit 13 (D.I. 112-18) at 1. Again, Mr. Pickett's coined term "association" referred to the operating system's "process of creating <u>dynamic links</u>," which "resolves <u>every</u>

[35] *See, e.g.*, JCC Appendix B, Exhibit 7 (D.I. 112-12 ('204 FH Paper No. 11, Appellant's Br.)) at 10.

intermodule reference" (emphases added). He then described step by step the process of "association":



Association:

When a dynamic link to module is created (its name and location are stored somewhere that any module can access to get to it), the reference list is also examined. Every single reference in the list is resolved through creation of implicit dynamic links, and the process is recursive. The recursion stops when a given module has no unresolved references, and the program is fully associated when the dynamic linking falls out of the recursion altogether. Every module that the program will ever need is now accessible through the dynamic links, and the OS holds on to these links, keeping track of where the modules are, until execution is complete.

JCC Appendix B, Exhibit 12 (D.I. 112-17) at 2. The intrinsic and extrinsic picture is consistent: "association" was a term Mr. Pickett employed to refer to a specific process in the prior art, one that created tables of "dynamic links" for "every single reference in the list" and through which "[e]very module that the program will ever need is now accessible." Microsoft's construction should be adopted because it incorporates these concepts and because it is straightforward and will be easy for the jury to understand.

2. <u>Terms governed by 35 U.S.C. § 112 ¶ 6</u>

The parties agree that four out of the five following terms are governed by 35 U.S.C. § 112 ¶ 6. Microsoft contends that the fifth term, "associator that forms an association based on said response," is so governed as well.

A claim element is considered a means-plus-function limitation under 35 U.S.C. § 112 ¶ 6 "when it is apparent that the element invokes purely functional terms, without the additional recital of specific structure or material for performing that function."[36] A claim that does not contain the term "means" is presumptively not subject to § 112 ¶ 6.[37] The presumption may be overcome, however, by showing that the claim term fails to recite sufficient structure or recites a function without reciting sufficient structure for performing that function.[38] The threshold issue

[36] *Al-site Corp. v. VSI Int'l, Inc.*, 174 F.3d 1308, 1318 (Fed. Cir. 1999).
[37] *MIT v. Abacus Software*, 462 F.3d 1344, 1353 (Fed. Cir. 2006).
[38] *Id.*

is whether the term itself connotes sufficient structure to one of ordinary skill in the art to perform the functions identified by each limitation, or whether the claim term has simply been used as a substitute for the term "means for."[39]

The parties agree that the functions of the claimed "code server" are implemented in software. The Federal Circuit's case law on software-implemented means-plus-function claims is well-developed and unambiguous—such claims must be supported by disclosure of an algorithm for carrying out the claimed function. "In a means-plus-function claim in which the disclosed structure is a computer, or microprocessor, programmed to carry out an algorithm, the disclosed structure is not the general purpose computer, but rather the special purpose computer programmed to perform the disclosed algorithm."[40] Disclosure of a general purpose computer, but not an algorithm to carry out the claimed function, is not enough:

> Because general purpose computers can be programmed to perform very different tasks in very different ways, simply disclosing a computer as the structure designated to perform a particular function does not limit the scope of the claim to 'the corresponding structure, material, or acts' that perform the function, as required by section 112 paragraph 6.[41]

While the sufficiency of an algorithm may be determined by one of ordinary skill in the art, some algorithm must be disclosed.[42] Without the disclosure of an algorithm, it is insufficient "to state or later argue that persons of ordinary skill in the art would know what structures to use to accomplish the claimed function."[43]

---

[39] *Apex Inc. v. Raritan Computer, Inc.*, 325 F.3d 1364, 1373 (Fed. Cir. 2003); *Lighting World, Inc. v. Birchwood Lighting, Inc.*, 382 F.3d 1354, 1360 (Fed. Cir. 2004).
[40] *WMS Gaming, Inc. v. Int'l Game Tech.*, 184 F.3d 1339, 1349 (Fed. Cir. 1999).
[41] *Aristocrat Techs. Austl. Pty. Ltd. v. Int'l Game Tech.*, 521 F.3d 1328, 1333 (Fed. Cir. 2008).
[42] *Id.* at 1337.
[43] *Id.*; *see also Harris Corp. v. Ericsson Inc.*, 417 F.3d 1241, 1243 (Fed. Cir. 2005); *Default Proof Credit Card Sys., Inc. v. Home Depot U.S.A.*, 412 F.3d 1291, 1302 (Fed. Cir. 2005); *Med. Instr. & Diag. Corp. v. Elekta AB*, 344 F.3d 1205, 1211-12 (Fed. Cir. 2003).

a. "associator that forms an association based on said response"

The parties disagree on whether this term should be construed under 35 U.S.C. § 112 ¶ 6. Microsoft contends it should. As an initial matter, it is instructive to look side-by-side at claim 9 of the '936 patent and claim 9 in the parent '204 patent:

Claim 9 of '204 Patent	Claim 9 of '936 Patent
said code server comprising: *** (d) **means for forming an association** based upon said protocol response.	said code server comprising: *** (d) **associator that forms an association** based upon said response.

Two things are clear: First, claim 9 of the '204 patent is presumptively governed by 112 ¶ 6. Second, the patentee made superficial linguistic changes to claim 9 of the '936 patent (changing "means for forming" to "associator that forms") in an attempt to remove it from the ambit of 112 ¶ 6. The problem with that approach is that "associator" does not have a particular meaning to a person skilled in the art.[44] Mr. Pickett coined the term. That makes sense, because as set forth in detail above, he also coined the term "forms an association." However, he did not define the term in the specification. Nor did he implicitly define the term, as REC likely will argue, because of the inescapable fact that claim 9 of the '936 patent, on its face, requires the code server to comprise the claimed "associator." Nowhere does the specification disclose a portion of the code server that "forms an association." In fact, as set forth in detail with respect to the "forming an association" terms, the patentee uniformly disclosed the code server's role as providing information to the "user" (operating system) so that the "user" could "form an association" in exactly the same way it did without a code server.

Because the word "associator" does not in and of itself connote structure and the specification does not disclose any structure in the code server that could correspond to the "associator," it is clear that "associator" is a placeholder that takes the place of "means," and thus insufficient to avoid treatment under 112 ¶ 6.[45] REC presumably understands this due to its preemptive construction in the alternative, in which is recites an alleged function (essentially REC's construction of "forming an association") and alleged corresponding structure.

REC's next problem, however, is that because the claimed code server is implemented in software, the corresponding structure must be an algorithm wherein the code server "forms an association." No such algorithm is disclosed. Microsoft contends that REC's plethora of citations do not disclose an algorithm to perform the claimed function at all, much less an algorithm that is performed by the code server. Failure to disclose an algorithm sufficient to perform the function renders a computer-implemented means-plus-function term indefinite.[46]

b. "means for searching for said discrete module in a storage area"

The parties' dispute concerns the algorithm disclosed in the specification corresponding to the function of "searching for said discrete module in a storage area." Microsoft has included portions of the specification that appear to describe how the "code module searcher" performs its search for discrete modules, including: (1) the default locations in which it searches; (2) the searching method; and (3) the ability to perform custom searches on a per-user basis.

The specification further states that the searching method "is the conventional method that would ordinarily be undertaken by the operating system in the absence of a code server." '936 patent at 4:47-64. The specification describes that conventional method as follows:

> Multi-module systems such as OS/2 and Windows have a set rule for locating the required code modules by searching through local directories and directories attached through a network to locate the code modules and extract the needed code module information. Under

[44] JCC Appendix B, Exhibit 3 (D.I. 112-8 (Morrissey Decl.)) at ¶ 40.
[45] *See, e.g., Lighting World*, 382 F.3d 1354 at 1360.
[46] *See, e.g., Blackboard, Inc. v. Desire2Learn, Inc.*, 574 F. 3d 1371, 1385 (Fed. Cir. 2009).

> OS/2 this is called LIBPATH and is called PATH under both Windows and DOS.[47]

The specification states that the searching method "is" the above-described conventional method, not that it "may be" or that a given embodiment performs such a method. Thus, the searching method is properly included in the algorithm because the patentee himself declared it essential to the function.

REC's construction is wrong because it seeks to incorporate limitations not disclosed in the specification as necessary to perform the function ("receiving a request for a discrete module"), and essentially rewrites the function in the remainder of its construction ("searching for the module in…a storage area").

c. "translation means for managing a translation of said discrete module from one form to another"

The parties' dispute boils down to whether the patent discloses any structure for *managing* a translation. Here, the specification fails to disclose any structure for the "managing" functionality.[48] Instead the specification teaches that "the *code server* is to *manage* the translation of code from one form to another." '936 patent at 5:65-67 (emphases added). That is, there must be some disclosed structure inside the code server that is responsible for *managing* the translation. The specification also teaches that while the "the code server *could* do the translations, it is preferable that the code server *simply oversee the translations done by other programs*." *Id.* at 6:6-8 (emphases added). In other words, performing a translation (i.e. means for translating) is different than overseeing the process done by another program (i.e. means for managing the translation). And yet, despite the distinction between actual translation and

[47] '936 patent at 1:28-34.
[48] REC's own claim construction points to the "translator" in the specification. But the claim is not "means for translating" but "means for *managing* a translation." REC's own purported construction demonstrates that the specification lacks structural support for this means element.

overseeing the process explained by patentee, there is no disclosed algorithm or structure for how the code server oversees (i.e. *manages*) this process.

Instead, the entirety of the teaching in the specification relating to the "translation" is contained in the following single sentence: "If the information table 24 does not contain code module information as to translation status and if the user requires translated code, at Block 30 the module is translated." *Id.* at 6:24-26. The patent does not discuss how the code server *manages* the process of translating a discrete module from one form to another. There is no further discussion of Block 30,[49] how the code server interacts with the external program that is performing the translation, or what actions must be taken in order to manage the process.[50] In short, there is absolutely no support in the specification for any structure associated with "*managing* a translation."

d. "means for managing a transfer of said discrete module to said first program"

Similar to the "means for managing a translation" term, the parties dispute centers around whether the specification discloses any structure for *managing* a transfer. Again, the specification fails to disclose any structure for the "managing" functionality.

The only disclosure in the specification relates to the actual *transfer* of code modules, and not any process for *managing* that transfer. *Id.* at 7:12-20 ("the server then enables the user to execute the program by some means such as *sending* the files in response for each request for a piece of code, *notifying* the system that the user already has the code, *transferring* the files asynchronously, *sending* each module as the computer accesses it, *sending* code as needed,

[49] Instead, the specification discloses a decision point in the code server of whether a translation is *needed*, but not the process of *managing* that translation. '936 patent at 6:11-21.

[50] For example, there is no discussion of what happens if there is an error in the translation process, if the external program cannot complete the translation, or any tasks that the code server must do in order to adequately "oversee" the process.

executing code via remote procedure calls, or *informing* the user as to other locations where the required code modules can be located.") None of the actions taught by the specification relates to the process of "*managing* a transfer"; instead they relate either the actual transfer itself ("sending" and "transferring"), to notifying the first program *where code can be found*, but not actually transferring code at all ("notifying" and "informing"), or to remote code execution without any transfer as well ("executing code via remote procedure call"). In other words, the specification fails to disclose any managerial functionality or structure for the process of performing a transfer.

e. "comparator means for comparing said request with said module information"

The parties dispute whether the specification discloses an algorithm for the function "comparing said request with said module information." In this case, the patent actually does disclose a portion of the code server responsible for comparing the request with the module information. However, no specific algorithm is disclosed. The patent merely says, "The output of the code module information table 24 is connected to a comparator 26 <u>which can be any device or software subroutine used for comparing the output of the code module information table 24 to the received protocol</u>." *Id.* at 3:51-55. In other words, REC seeks to capture every form whatsoever of comparing two pieces of information. This is pure functional claiming, which is improper—the specification must disclose some <u>specific</u> algorithm.[51]

REC's construction is incorrect, not only because it does not even claim to reduce the scope of the claim to a specific algorithm, but because it rewrites the function using four times the number of words. Indeed, its third step alone, "determining whether the requested module

[51] *See, e.g., Encyclopedia Britannica, Inc. v. Alpine Elecs., Inc.*, 355 Fed. Appx. 389 (Fed. Cir. 2009).

information corresponds to module information from the module information table," is essentially the same as the function. REC's construction thus is improper and should be rejected.

IV. CONCLUSION

For the above reasons, Microsoft respectfully requests that the Court adopt its proposed claim constructions.

DATED this 22nd day of November, 2011.

DANIELSON HARRIGAN LEYH &
TOLLEFSON LLP

By s/Shane P. Cramer
Arthur W. Harrigan, Jr., WSBA #1751
Christopher Wion, WSBA #33207
Shane P. Cramer, WSBA #35099

Isabella Fu
MICROSOFT CORPORATION
1 Microsoft Way
Redmond, WA 98052
Phone: 425-882-8080
Fax: 425-708-1507

Ruffin B. Cordell (*pro hac vice*)
cordell@fr.com
Joshua B. Pond (*pro hac vice*)
pond@fr.com
1425 K Street N.W., 11th Floor
Washington, DC 20005
Telephone: 202-783-5070
Facsimile: 202-783-2331

Katherine K. Lutton (*pro hac vice*)
lutton@fr.com
Kelly C. Hunsaker (*pro hac vice*)
Hunsaker@fr.com
Scott A. Penner (*pro hac vice*)
Penner@fr.com
500 Arguello Street, Suite 500
Redwood City, CA 94063
Telephone: 650-839-5070
Facsimile: 650-839-5071

Benjamin C. Elacqua *(pro hac vice)*
elacqua@fr.com
Brian G. Strand *(pro hac vice)*
strand@fr.com
One Houston Center
1221 McKinney Street, Suite 2800
Houston, TX 77010
Telephone: 713-654-5300
Facsimile: 713-652-0109

Counsel for Defendant Microsoft Corp.

CERTIFICATE OF SERVICE

I hereby certify that on November 22, 2011, I electronically filed the foregoing with the Clerk of the Court using the CM/ECF system which will send notification to all counsel of record who are deemed to have consented to electronic service.

s/Shane P. Cramer
Shane P. Cramer, WSBA #35099
Attorneys for Defendant
Microsoft Corporation